STRING ATLAS: BREAKING THE ONE-TOKENIZER ASSUMPTION IN LANGUAGE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

A standard language model is trained with a single canonical tokenizer, so that tokenizer is effectively frozen into its weights. This structural constraint means that changing the tokenizer afterwards invalidates the learned embedding parameters connecting raw text to the Transformer trunk. Yet, tokenizer choice is far from a cosmetic preprocessing step; it dictates domain compression, effective context length, arithmetic capabilities, and multilingual fairness. While existing methods either commit to one universal vocabulary upfront or resort to expensive post-hoc vocabulary transfer, we introduce String Atlas, an architecture that natively supports multiple tokenizers from pretraining. Instead of equipping each tokenizer with private embedding tables, which fragments learning, String Atlas keys its lexical parameters directly to the emitted byte string. Tokenizers may disagree on boundaries, but they seamlessly share rows in a common atlas whenever they emit the same string. Across suites of tokenizer sets varying domains, granularity, and even algorithm, we find that String Atlas allows a language model to support several tokenizer interfaces simultaneously, nearly matching the performance of independently trained specialists while maintaining the training and inference costs of a single model. Together, these results break the one-tokenizer assumption that binds a model to its vocabulary and reframe tokenizers as interchangeable views over a single shared lexicon.

1 INTRODUCTION

A tokenizer is the interface through which a language model (LM) sees text; it segments a stream of bytes into a sequence of discrete tokens in the LM's vocabulary (Sennrich et al., 2016; Kudo, 2018). A model is trained to read and predict tokens in exactly this vocabulary, so the input embeddings and output projection are defined over that vocabulary alone. The tokenizer, in effect, becomes part of the model.

This convention is so universal that the tokenizer is rarely treated as a choice at all, yet the choice of tokenizer often has large downstream effects on the trained model. Which tokenizer a model uses changes how well it does arithmetic (Singh & Strouse, 2024), how efficiently it handles languages other than English (Petrov et al., 2023), how it manipulates text at the character level (Zheng et al., 2025), and how well it compresses text from different domains (Bommarito et al., 2025; Dagan et al., 2024; Herold et al., 2025). For example, a segmentation that compresses code well might not be one that compresses prose or mathematics well, and a granularity that is necessary for one task can be wasteful for another, so the choices a model would benefit from often conflict.

At the same time, trained models are effectively locked to the single tokenizer they were built around. One can train a separate model (i.e., a "specialist") for each tokenizer, or apply a post-hoc tokenizer-transfer method that re-initializes the lexical parameters and continues training (Minixhofer et al., 2022; Dobler & de Melo, 2023; Minixhofer et al., 2024; Nakash et al., 2025; Liyanage & Yvon, 2026), usually with some performance degradation. Although these are useful, complementary tools, they do not give one subword model several native tokenizer interfaces from the start.

In this work, we ask whether one LM can support many tokenizers *natively*, from pretraining. A direct approach would be to keep one Transformer trunk and give each tokenizer its own embedding and unembedding tables, a baseline we call *Private Tables* (Figure 1b). This makes the tokenizer interfaces available, but it makes them unrelated to one another. When two tokenizers both emit the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

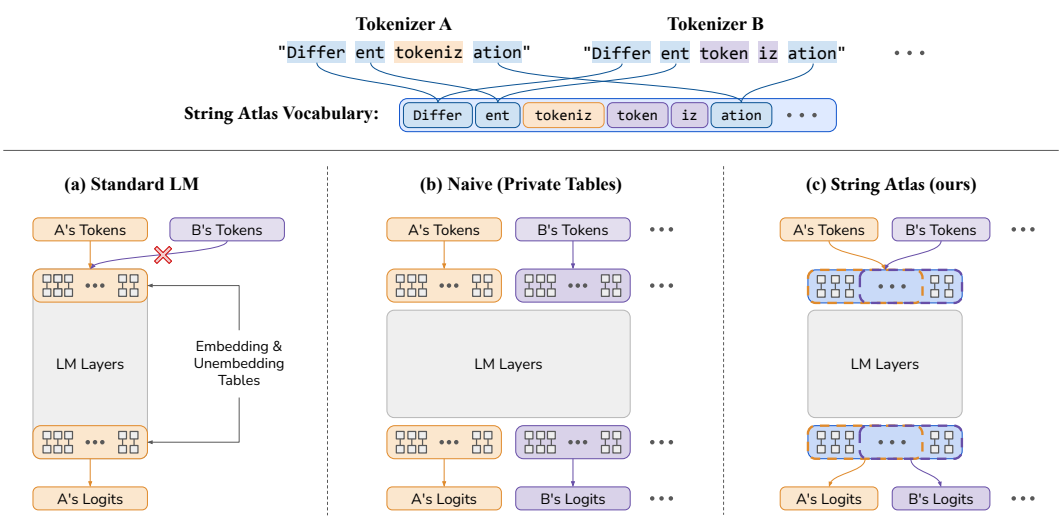


Figure 1: The one-tokenizer assumption and how String Atlas relaxes it. **(a)** A standard language model ties its learned token parameters to one tokenizer's ids. **(b)** The naive multi-tokenizer model keeps a private table per tokenizer, so a string shared by two tokenizers (e.g., "`ation`") is stored as two unrelated rows. **(c)** String Atlas keys the parameters by the emitted byte string, so shared strings collapse to a single row and each tokenizer becomes a view over one common atlas.

byte string "`ation`", Private Tables stores two independent rows for it and learns each only from the occurrences seen under its own tokenizer, so the evidence for one lexical object is split across tokenizer-local ids. This leads to substantial performance degradations (Section 5).

We introduce **String Atlas** to fix this ownership problem. Even when tokenizers split text differently, many of their tokens represent the same strings. String Atlas keys token parameters by byte string rather than by a tokenizer-specific id. Every tokenizer indexes into one shared table, so a string several tokenizers have in common is learned once and accumulates evidence from all of them, while each tokenizer remains an ordinary segmentation rule with an ordinary next-token softmax over its own vocabulary (Figure 1c). The trunk, objective, and decoding procedure are unchanged.

Our experiments support three main claims. First, we demonstrate that a single language model can natively support many tokenizers with little loss in quality. String Atlas nearly matches a fleet of independently trained per-tokenizer specialists while only requiring the training compute of one specialist. Second, the strength of our proposed model stems from pooling supervision on shared rows; Private Tables shares the trunk but not the strings, and it does not close the gap; once per-row training exposure is matched, the rows String Atlas learns nearly match a specialist's. Third, this pooling requires training under the tokenizers jointly and is not a consequence of overlapping vocabularies alone. Copying a specialist's rows onto a new tokenizer by byte-string match leaves a large quality gap even when the copied rows cover $>99\%$ of evaluated positions, because a row trained under one tokenizer's segmentation is miscalibrated for another's, and String Atlas closes that gap precisely because it trains every shared row under all the tokenizers at once.

2 Tokenizer Variation as a Modeling Target

If no single tokenizer is best for everything, then which kinds of variation are worth keeping? We study three representative design choices in this paper: tokenizers can differ in the domain they are tuned for, their granularity, and in the algorithm that produced them. The three make increasingly demanding tests of the same idea: the first varies only the data behind a fixed recipe, the second varies vocabulary size, and the third varies the segmentation procedure itself.

Table 1: Different tokenizers compress different domains best. Each column is one of the five model-training tokenizers, and each row is an evaluation domain. Entries are bytes per token measured on 2 MiB of held-out SlimPajama validation/test text from that domain. Higher is better, darker shading indicates better compression within each row, and bold marks the best tokenizer for that domain.

	Tokenizer (bytes/token ↑)				
Eval domain	**General**	**Clean web**	**Code skew**	**Sci/math**	**Wiki/QA**
CommonCrawl	**4.208**	4.187	4.131	4.139	4.118
C4	4.444	**4.458**	4.385	4.392	4.362
GitHub	3.099	2.971	**3.292**	2.782	3.048
StackExchange	3.275	3.252	**3.416**	3.093	3.312
arXiv	3.049	2.825	2.931	**3.169**	2.897
Wikipedia	2.974	3.215	2.818	3.331	**3.498**

Domain. A tokenizer trained on one kind of text allocates its vocabulary to the most frequent patterns in that text, which can reduce token counts, improve effective context length, and lower deployment cost on matching domains (Dagan et al., 2024; Bommarito et al., 2025; Herold et al., 2025). To make this axis concrete, we build five byte-level BPE tokenizers with 32k vocabulary, one for each of five domain mixtures: a general mixture, cleaned web, code-skewed text, science and mathematics, and Wikipedia with QA (Figure 2). We sample that text from the SlimPajama dataset (Soboleva et al., 2023), which labels every document with its source domain, and train each tokenizer on 512 MiB drawn from its own mixture. These mixtures are used only for tokenizer training; the language models are trained and evaluated on a separate matched text stream. Table 1 shows that the five tokenizers specialize in predictable ways: the cleaned-web tokenizer compresses C4 best, the code-skewed one GitHub and StackExchange, the science tokenizer arXiv, and the Wikipedia/QA tokenizer Wikipedia. This five-tokenizer set is our main testbed for the rest of the paper.

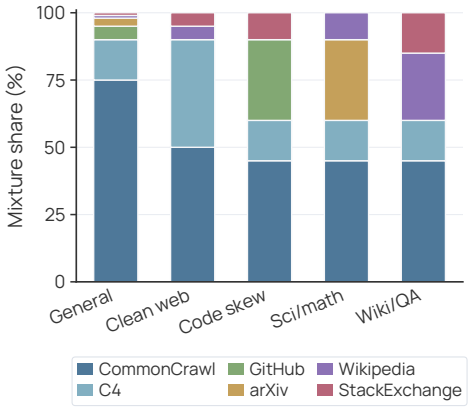


Figure 2: Tokenizer-training mixtures for the five domain-specific BPE tokenizers.

Granularity. Vocabulary size sets the granularity of the tokens. Prior work suggests that this choice interacts with both scaling and downstream behavior: larger models may prefer larger vocabularies, compression tends to correlate with performance, but token count alone does not determine tokenizer quality (Tao et al., 2024; Goldman et al., 2024; Schmidt et al., 2024). A small vocabulary breaks text into many short, compositional pieces, which can help expose spelling, character, and number structure, while a large one uses fewer, longer tokens that are closer to whole words or domain-specific atoms. Stochastic segmentation methods exploit this same axis by training on multiple subword decompositions of the same text (Kudo, 2018; Provilkov et al., 2020; Sims et al., 2026). The right granularity often depends on what the model is being asked to do. We instantiate this axis with BPE tokenizers of vocab size 16k, 32k, and 64k (Section 6).

Algorithm. Tokenizers built by different algorithms segment text differently even at the same vocabulary size, because BPE, WordPiece, and Unigram learn and apply their vocabularies in different ways (Sennrich et al., 2016; Kudo, 2018). This algorithmic choice reaches final model quality. Unigram tokenization has been shown to match or outperform BPE for language-model pretraining, and formal and empirical analyses of BPE point to limitations of its greedy merge procedure (Bostrom & Durrett, 2020; Zouhar et al., 2023; Chizhov et al., 2024). Supporting several algorithms is also a harder test than supporting several domain-specific BPE tokenizers, since the views now share neither a construction rule nor a segmentation procedure.

Taken together, the three axes point in one direction. No single tokenizer is the right interface for every input, and which one is right depends on the text at hand. This motivates native multi-tokenizer support as a modeling primitive: before a system can select among tokenizer routes, those routes must first coexist inside one model, remain calibrated under their own next-token distributions, and avoid the cost of training a separate specialist for each one. We now study this primitive directly.

3 PRELIMINARIES

Tokenizer-bound language models. A standard language model interacts with text through a tokenizer T, which maps raw byte strings into a fixed vocabulary V of discrete IDs. Each ID $i \in V$ is assigned its own distinct "row" in the model's parameters, consisting of an embedding vector $E[i]$, an unembedding vector $U[i]$, and a token bias $b[i]$. During a forward pass over a sequence $i_1, \ldots, i_n$, the Transformer trunk ingests these embeddings to generate hidden states h_t, and the model computes the next-token logits $h_t^\top U[j] + b[j]$ across all $j \in V$.

Measuring quality across tokenizers. Different tokenizers cut the same text into different numbers of tokens, so token-level perplexity is not comparable between them. Throughout the paper we report *bits per byte* (BPB), the model's total negative log-likelihood in base two divided by the number of underlying bytes,

$$\mathrm{BPB} \;=\; -\frac{1}{|\mathrm{bytes}|}\sum_t \log_2 p(i_{t+1} \mid i_{\le t}). \tag{1}$$

Because the denominator is bytes rather than tokens, BPB lets us compare model quality under different segmentations on equal footing.

4 METHOD: STRING ATLAS

Setup and notation. Suppose we want one model to support R tokenizers $T_1, \ldots, T_R$, with vocabularies $V_1, \ldots, V_R$. For tokenizer T_r with vocabulary V_r, let D_r denote its detokenization function. Then for a token id $i \in V_r$, $D_r([i])$ is the byte string emitted by that token.

Private Tables. A straightforward baseline is to give each tokenizer its own input and output tables, so token i under tokenizer T_r has embedding vector $E_r[i]$, unembedding vector $U_r[i]$, and token bias $b_r[i]$. However, this means that identical strings are still treated as unrelated tokens. If $i \in V_r$ and $i' \in V_{r'}$ both represent the same byte string (i.e., $D_r([i]) = D_{r'}([i'])$) they nevertheless have separate parameters allocated within the model, and the two copies are learned independently.

Indexing by string instead of by id. String Atlas instead gives each *distinct byte string* a single embedding vector, unembedding vector, and bias, shared by every tokenizer that contains that string. To do this, we first form the *atlas*, the set of all byte strings represented by any tokenizer,

$$S \;=\; \bigcup_{r=1}^{R} D_r([i]) : i \in V_r\,, \tag{2}$$

and define a lookup $a_r(i) = \mathrm{id}\big(D_r([i])\big) \in \{1, \ldots, |S|\}$ that maps each tokenizer-specific id to the corresponding row in the atlas. We then learn one set of tables $E, U \in \mathbb{R}^{|S|\times d}$ and $b \in \mathbb{R}^{|S|}$ over these shared rows. Thus, if two tokenizers contain ids for the same byte string, both ids point to the same E, U, and b rows in the shared atlas.

The tokenizer itself otherwise works as usual. Under tokenizer T_r, a token i_t is embedded as $x_t = E[a_r(i_t)]$, and after the model produces hidden state h_t, the next-token distribution is computed over that tokenizer's own vocabulary:

$$p_r(j \mid i_{\le t}) \;=\; \frac{\exp\big(h_t^\top U[a_r(j)] + b[a_r(j)]\big)}{\sum_{k\in V_r}\exp\big(h_t^\top U[a_r(k)] + b[a_r(k)]\big)}, \qquad j \in V_r. \tag{3}$$

In other words, each tokenizer keeps its own segmentation and next-token vocabulary; the only difference is that tokens representing the same byte string now share the same learned rows.

Training. We spread training uniformly over the pool of tokenizers; for each microbatch, we sample a tokenizer route r, draw raw text windows from the shared training stream, tokenize them with T_r, and minimize ordinary next-token cross-entropy over T_r's vocabulary. If $\mathrm{count}_r(s)$ is how often string s appears as a target under tokenizer T_r, then Private Tables trains a tokenizer-local row on $\mathrm{count}_r(s)$ occurrences, whereas String Atlas trains one shared row on $\sum_r \mathrm{count}_r(s)$ occurrences.

Parameter savings. In addition to pooling supervision, sharing rows reduces the number of lexical parameters. Independently trained specialists require R trunks and R tokenizer-specific embedding/unembedding tables; Private Tables requires one trunk and still R tokenizer-specific tables; String Atlas requires one trunk and just one table over S:

$$P_{\text{spec}} \approx R\,P_{\text{trunk}} + \textstyle\sum_r (2d{+}1)|V_r|, \quad P_{\text{priv}} \approx P_{\text{trunk}} + \textstyle\sum_r (2d{+}1)|V_r|, \quad P_{\text{atlas}} \approx P_{\text{trunk}} + (2d{+}1)|S|. \tag{4}$$

The saving is governed by how much the vocabularies overlap, since $|S| \le \sum_r |V_r|$ with equality only if no two tokenizers share a string. At inference the model evaluates one tokenizer at a time over its own vocabulary, so a forward pass costs what it would for a standard model with that active tokenizer.

Why not merge the tokenizer vocabularies. A seemingly simpler alternative to multi-tokenizer support is to combine all of the tokenizer vocabularies into one larger vocabulary and train a standard language model with it. We test this as the Union-Longest baseline in Section 5. Union-Longest uses exactly the same set of byte strings as the String Atlas: we take the union of the strings from all tokenizer vocabularies, and then tokenize text greedily by selecting the longest vocabulary string that matches at each position. This produces one deterministic tokenizer and discards the original BPE segmentation rules. To test whether that single merged tokenizer allows the model to support the original routes, we evaluate the trained model under each original tokenizer by mapping its tokens to the corresponding union rows and restricting the output vocabulary to that tokenizer's entries. See Appendix D for more details.

5 Main results

Setup. We evaluate on the five domain-specific BPE tokenizers from Section 2. We compare five configurations and note their relative cost to support R tokenizers:

- **Specialists (full)** ($R\times$ **compute**): one full model trained per tokenizer (the quality ceiling).
- **Specialists (CM)** ($1\times$ **compute**): the same per-tokenizer specialists, but **c**ompute-**m**atched to a single shared-model run, so each is trained on $\frac{1}{R}$ of the full specialist budget.
- **Private Tables** ($1\times$ **compute**): one Transformer trunk shared across tokenizers, with each tokenizer keeping its own private E_r, U_r, b_r.
- **Union-Longest** ($1\times$ **compute**): one ordinary model trained on the union of all emitted strings, using a deterministic longest-match tokenizer over that union. At evaluation, we restrict the active vocabulary to each original tokenizer route.
- **String Atlas** ($1\times$ **compute**): one shared Transformer trunk with lexical parameters keyed by emitted byte string, so any string shared by multiple tokenizers uses the same row.

Every configuration here uses the same 86.6M-parameter trunk architecture and the same training and evaluation stream. We train each model on 1B tokens (8k optimizer steps) of the FineWeb-Edu corpus (Penedo et al., 2024). Multi-tokenizer runs split those positions approximately uniformly across the five tokenizer routes as described in Section 4; Union-Longest instead spends the same budget on its single longest-match route; the five-specialist ceiling uses five separate runs. We report BPB on each of the five original tokenizer routes and the mean across them in Table 2 and Table 3.

Main result. At a one-model budget, String Atlas nearly matches the full five-specialist ceiling across every tokenizer route, with a mean gap of only $+0.0137$ BPB. The other one-model baselines are far behind. Most notably, Union-Longest has access to the same emitted strings as String Atlas, but training those strings under a single longest-match policy does not calibrate the model for the

Table 2: Main result on five domain-specific BPE tokenizers. Specialists (full) are the five-run ceiling; all other configurations use a one-model training budget. Gap is to the full-specialist ceiling; lower BPB is better.

	Configuration			Performance	
Method	Total params	Lexical rows	Training FLOPs	Mean BPB (↓)	Gap (↓)
Specialists (full)	679.1M	160k	$4.270 \cdot 10^{18}$ $(5\times)$	1.0890	–
Specialists (CM)	679.1M	160k	$0.852 \cdot 10^{18}$ $(1\times)$	1.2881	+0.1991
Private Tables	332.5M	160k	$0.854 \cdot 10^{18}$ $(1\times)$	1.3065	+0.2175
Union-Longest	163.8M	50.2k	$0.852 \cdot 10^{18}$ $(1\times)$	1.2835	+0.1945
String Atlas (ours)	163.8M	50.2k	$0.852 \cdot 10^{18}$ $(1\times)$	**1.1027**	**+0.0137**

Table 3: Per-route gaps relative to full specialists. The first row gives full-specialist BPB; remaining rows report the BPB tax relative to the full specialist on the same tokenizer route. Lower is better.

	Per-tokenizer performance					
Method	General	Clean web	Code skew	Sci/math	Wiki/QA	**Average** (↓)
Specialists (full) [BPB]	1.0888	1.0822	1.0905	1.0911	1.0922	1.0890
Specialists (CM)	+0.1952	+0.1994	+0.2037	+0.1974	+0.2002	+0.1992
Private Tables	+0.2161	+0.2179	+0.2171	+0.2146	+0.2218	+0.2175
Union-Longest	+0.1634	+0.1688	+0.2137	+0.1994	+0.2274	+0.1945
String Atlas (ours)	**+0.0101**	**+0.0165**	**+0.0155**	**+0.0127**	**+0.0138**	**+0.0137**

original BPE routes, yielding a $+0.1945$ BPB penalty when evaluated through them. String Atlas avoids this collapse by preserving each tokenizer's own next-token route while sharing rows only when routes emit the same byte string. Additional Union-Longest analyses are in Appendix D.

Results at larger scales. The main experiments use an 86.6M-trunk setting. In Appendix E, we also train larger 500M and 1B trunks at Chinchilla-style token budgets (Hoffmann et al., 2022) to check whether the pattern persists beyond the small-model regime. We find that String Atlas remains close to specialist quality at both scales.

Post-hoc transfer and downstream evaluation. Appendix F compares against FOCUS (Dobler & de Melo, 2023), a strong post-hoc tokenizer-transfer baseline, and reports results on downstream multiple-choice benchmarks. Across model scales, String Atlas remains substantially closer to the full specialists in likelihood than zero-shot FOCUS, while also closely matching specialist performance on the downstream evaluations.

6 Granularity and algorithm

The domain tokenizers differ only in their training data. We next test harder settings where tokenizers also differ in granularity or segmentation algorithm.

We first vary granularity while keeping the tokenization algorithm fixed. We use BPE vocabularies of 16k, 32k, and 64k in two settings. In the *nested* suite, all three tokenizers come from the same merge sequence, so the smaller vocabularies are exact prefixes of the larger ones. This is also the setting most favorable to stochastic segmentation: methods such as StochasTok (Sims et al., 2026) expose one model to different coarse and fine segmentations within a single aligned vocabulary. We therefore compare against StochasTok at its default rate $p = 0.1$. In the *independent* suite, we train 16k, 32k, and 64k tokenizers on separate sets of training data, so their vocabularies no longer nest.

We next vary the tokenization algorithm itself. Here the vocabulary size and training data are held fixed, but the three routes use BPE, WordPiece, and Unigram tokenizers with 32k entries. This removes the common BPE construction rule entirely and tests whether String Atlas still works when the routes differ in how they build and select tokens.

Table 4: String Atlas stays near the specialist ceiling on every axis. Gap is to the full-specialist ceiling within each suite; lower BPB is better.

Suite	Method	Mean BPB	Gap	Rows stored
Nested 16k/32k/64k	Specialists (full)	1.0903	–	112k
	Specialists (CM)	1.2003	+0.1100	112k
	Private Tables	1.2243	+0.1340	112k
	Union-Longest	1.2028	+0.1125	64k
	StochasTok	1.1789	+0.0886	64k
	String Atlas (ours)	**1.1088**	**+0.0185**	**64k**
Independent 16k/32k/64k	Specialists (full)	1.0879	–	112k
	Specialists (CM)	1.1984	+0.1105	112k
	Private Tables	1.2176	+0.1297	112k
	Union-Longest	1.3221	+0.2342	65.1k
	String Atlas (ours)	**1.1098**	**+0.0219**	**65.1k**
BPE/WordPiece/Unigram 32k	Specialists (full)	1.0884	–	96k
	Specialists (CM)	1.1989	+0.1105	96k
	Private Tables	1.2153	+0.1269	96k
	Union-Longest	1.3540	+0.2656	57.3k
	String Atlas (ours)	**1.1152**	**+0.0268**	**57.3k**

Table 4 shows the same pattern across all three suites. String Atlas stays within $+0.03$ BPB of the full specialist ceiling everywhere and outperforms every one-model baseline. StochasTok helps in the nested setting, where its random segmentations remain inside one aligned vocabulary, but String Atlas reduces the gap further by training each tokenizer route directly. Overall, we find that the advantage persists when the vocabularies are trained independently and even when the tokenization algorithm itself changes. Per-route results can be found in Appendix B.

7 Analysis

The experiments above show that String Atlas can support several tokenizer routes with little loss in quality. We next isolate why, through ablations, row-supervision analysis, and zero-shot transfer.

7.1 Exact string identity matters

Table 5: Mechanism ablations on the five-tokenizer suite. Private Tables removes cross-tokenizer lexical sharing. Shuffled Atlas preserves the amount and structure of tying but randomizes the string-to-row mapping per tokenizer. Input-only and output-only variants isolate which side of the lexical interface benefits more from sharing.

Configuration	Input $\mathbf{E}$ key	Output $\mathbf{U}, \mathbf{b}$ key	Mean gap
Private Tables	tokenizer id	tokenizer id	+0.2175
Shuffled Atlas	shuffled string	shuffled string	+0.2718
Input-only String Atlas	string	tokenizer id	+0.0690
Output-only String Atlas	tokenizer id	string	+0.0284
Full String Atlas	string	string	**+0.0137**

Private Tables is the direct no-sharing ablation; if the main benefit came from exposing one trunk to several tokenizers, Private Tables should be close to String Atlas. Instead it trails the specialist ceiling by $+0.2175$ BPB, while full String Atlas is only $+0.0137$ behind (Table 5). The shared trunk is therefore not the source of the gain.

The gain also does not come from tying rows arbitrarily. For Shuffled Atlas, we keep the same atlas size, the same active row set for each tokenizer, and the same route-level tying pattern as String Atlas, but we randomly permute the string-to-row assignment separately for each tokenizer. That

is, a row that represents "ation" under one tokenizer might represent "the" under another. Shuffled Atlas performs worse than Private Tables, with a $+0.2718$ BPB gap. Tying unrelated strings creates conflicting supervision for the model, so exact byte-string identity is what makes tying useful.

Finally, the input-only and output-only variants show where most of the benefit comes from. Sharing the output rows and biases recovers more quality than sharing the input embeddings alone. This points to the next-token prediction side as the main bottleneck (String Atlas works largely because several tokenizer routes pool target-event supervision onto the same output row).

7.2 Sharing pools row supervision

Across the five domain-specific BPE tokenizers, Private Tables stores 160k tokenizer-local rows, whereas String Atlas stores 50,184 shared atlas rows. Beyond just the reduction in parameters, this changes how much supervision each row receives. Figure 3 shows the basic pooling effect. The same frequent byte strings appear under several tokenizers, so String Atlas routes their target events into one row instead of splitting them across several tokenizer-local copies. In the five-tokenizer suite, the mean row exposure rises from roughly 350 target events per private-table row to 1,122 per atlas row, and the maximum row exposure rises from about 0.43M to 2.12M target events.

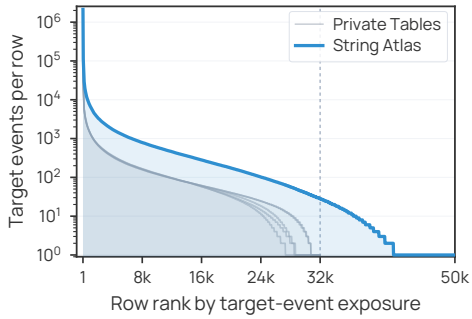


Figure 3: String Atlas pools target-event supervision onto shared string rows at matched training budgets. Rows are sorted by positive target-event exposure.

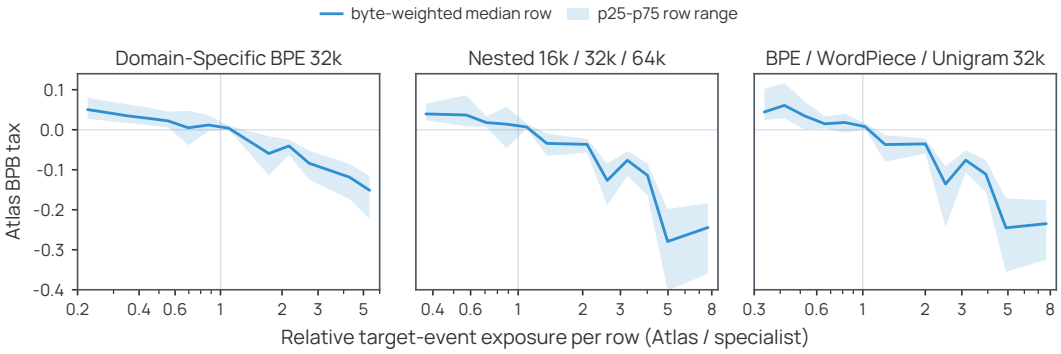


Figure 4: Row quality improves with target-event exposure. Rows are binned by their target-event exposure in String Atlas relative to the matching specialist, so $x = 1$ denotes equal exposure. The solid curve shows the byte-weighted median BPB difference between String Atlas and the specialist (negative values favor String Atlas). Shading shows the interquartile range across rows.

If pooled supervision is the mechanism, row quality should vary predictably with target-event exposure. Figure 4 tests this across the domain-mixture, nested-granularity, and algorithm suites by comparing shared String Atlas rows with the corresponding specialist row copies. We summarize the row-level BPB taxes using a median weighted by the evaluated bytes attributed to each row (this is the byte-level analogue of frequency weighting, and it accounts for the fact that tokens represent strings of different lengths). Most importantly, from the figure, we find that a transition occurs right around the matched-exposure point. Underexposed atlas rows have a small BPB tax, rows near one-to-one exposure are essentially tax-free, and rows with extra supervision move below zero (meaning the shared row is a better fit on positions targeting that string). This provides strong evidence that String Atlas's quality comes from pooled supervision on exact shared rows.

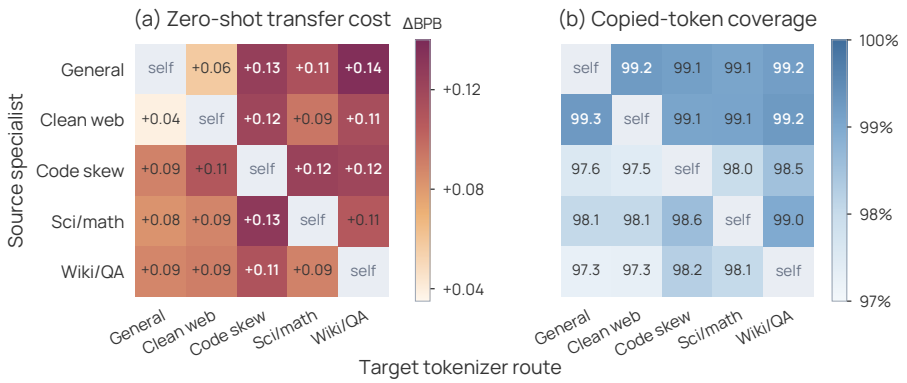


Figure 5: Zero-shot row transfer between the five domain-specific BPE specialists. Left: BPB increase relative to the target specialist. Right: copied target-position coverage before applying the contiguous-run filter. High copied-row coverage does not imply calibrated target-route quality.

7.3 Coverage is not enough without joint training

Because tokenizer vocabularies often overlap, a natural alternative to joint multi-tokenizer training is to train a specialist first and copy matching rows between tokenizers afterward. We test this directly by taking each source specialist, copying rows into each target tokenizer by exact byte-string match, and evaluating whether the resulting zero-shot target model can recover target-specialist quality. To make this a conservative test of row transfer, we only evaluate spans of eval text where the input context and target tokens are all covered by copied rows.

Figure 5 shows that high coverage is not enough. Off-diagonal copied target-position coverage ranges from 97.3% to 99.3%, and our covered-span filter still scores 94.6% to 98.5% of positions. Yet the zero-shot BPB increase remains large, averaging $+0.101$ BPB off diagonal and reaching $+0.136$ BPB. The copied rows are present, but they were trained under the source tokenizer's segmentation and softmax competition. String Atlas avoids this failure mode because the shared row is trained directly under every tokenizer route where it will be used. Appendix C also repeats this test with more heavily skewed domain tokenizers, and even though zero-shot transfer becomes substantially harder, jointly trained String Atlas remains close to the specialist ceiling.

8 Discussion

Throughout this work, we have shown that the one-tokenizer assumption is a structural artifact rather than a fundamental requirement of language modeling. When token parameters are tied to specific IDs, lexical knowledge is fragmented and the model relearns shared text in isolation. String Atlas shows that breaking this bottleneck requires nothing more than a change of index. By shifting the parameter key from the tokenizer ID to the underlying byte string, we effectively decouple the segmentation policy from the learned representations. The broader implication is that tokenizer design can become less brittle. Instead of committing all pretraining to one canonical tokenizer, a model can expose multiple calibrated views over the same learned lexicon, including domain-specific views, finer or coarser views, and even views produced by different tokenizer algorithms.

Limitations and future work. Our results establish native multi-tokenizer pretraining as a modeling primitive, but they do not by themselves determine the best downstream policy for choosing among tokenizer routes. That policy layer opens several promising directions, for example: learning route-selection rules that trade off likelihood, compression, latency, and task quality; using tokenizer-flexible models as interoperability layers across model families (e.g. as "universal" draft models for speculative decoding for a variety of target models); and treating tokenizer construction as a portfolio design problem, where the goal is not to find one globally optimal tokenizer but to choose a complementary set of views for different domains, granularities, scripts, and system constraints. These applications require task- and system-level objectives beyond route-level language modeling, so we leave them to future work.

AI USE STATEMENT

We used generative AI tools to assist with initial drafting and language editing of the manuscript. All AI-assisted text was reviewed and extensively revised by the authors, and all technical claims, citations, experiments, and reported results were checked by the authors. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Appendix G provides the details needed to reproduce our experiments, including tokenizer construction, model architectures and training hyperparameters, String Atlas construction, and the training and inference procedure. The main text and remaining appendices specify the evaluation protocols and report the corresponding route-level and ablation results.

REFERENCES

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9904–9923, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Michael J. Bommarito, II, Daniel Martin Katz, and Jillian Bommarito. KL3M tokenizers: A family of domain-specific and character-level tokenizers for legal, financial, and preprocessing applications, 2025.

Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4617–4624, 2020.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. BPE gets picky: Efficient vocabulary refinement during tokenizer training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16587–16604, 2024.

Hyung Won Chung, Dan Garrette, Kiat Chuan Tan, and Jason Riesa. Improving multilingual models with language-clustered vocabularies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4536–4546, 2020.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Øyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. Getting the most out of your tokenizer for pre-training and domain adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 9784–9805, 2024.

Konstantin Dobler and Gerard de Melo. FOCUS: Effective embedding initialization for monolingual specialization of multilingual models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13440–13454, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL `https://zenodo.org/records/10256836`.

Charles Goddard and Fernando Fernandes Neto. Training-free tokenizer transplantation via orthogonal matching pursuit, 2025.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2274–2286, 2024.

Christian Herold, Michael Kozielski, Nicholas Santavas, Yannick Versley, and Shahram Khadivi. Vocabulary customization for efficient domain-specific LLM deployment. In *NeurIPS 2025 Workshop on Continual and Compatible Foundation Model Updates (CCFM)*, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 66–75, 2018.

Tomasz Limisiewicz, Jiří Balhar, and David Mareček. Tokenization impacts multilingual language modeling: Assessing vocabulary allocation and overlap across languages. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 5661–5681, 2023.

Vijini Pilana Liyanage and François Yvon. AdaptBPE: From general purpose to specialized tokenizers. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2607–2620, 2026.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *CoRR*, abs/2112.10508, 2021.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3992–4006, 2022.

Benjamin Minixhofer, Edoardo M. Ponti, and Ivan Vulić. Zero-shot tokenizer transfer. In *Advances in Neural Information Processing Systems*, volume 37, pp. 46791–46818, 2024.

Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. Universal cross-tokenizer distillation via approximate likelihood matching. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Itay Nakash, Nitay Calderon, Eyal Ben David, Elad Hoffer, and Roi Reichart. AdaptiVocab: Enhancing LLM efficiency in focused domains through lightweight vocabulary adaptation. In *Proceedings of the Second Conference on Language Modeling*, 2025.

Malte Ostendorff and Georg Rehm. Efficient language model training through cross-lingual and progressive transfer learning. In *Proceedings of the PML4DC Workshop co-located with ICLR 2023*, 2023.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E. Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, 2025.

Vaidehi Patil, Partha Talukdar, and Sunita Sarawagi. Overlap-based vocabulary generation improves cross-lingual transfer among related languages. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 219–233, 2022.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849, 2024.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*, volume 36, pp. 36963–36990, 2023.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1882–1892, 2020.

François Remy, Pieter Delobelle, Hayastan Avetisyan, Alfiya Khabibullina, Miryam de Lhoneux, and Thomas Demeester. Trans-tokenization and cross-lingual vocabulary transfers: Language adaptation of LLMs for low-resource NLP. In *First Conference on Language Modeling*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 678–702, 2024.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, 2016.

Franck Signe, Hippolyte Pilchen, François Yvon, and Édouard Grave. To each language its tokenizer: Modular tokenizers for efficient multilingual LLMs. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026.

Anya Sims, Thomas Foster, T. Duy Nguyen-Hien, Klara Kaleb, Joseph Lee, Jakob Nicolaus Foerster, Yee Whye Teh, and Cong Lu. StochasTok: Improving fine-grained subword understanding in LLMs. In *International Conference on Learning Representations (ICLR)*, 2026.

Aaditya K. Singh and DJ Strouse. Tokenization counts: The impact of tokenization on arithmetic in frontier LLMs, 2024.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In *Advances in Neural Information Processing Systems*, volume 37, pp. 114147–114179, 2024.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations (ICLR)*, 2022.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Sarah Wiegreffe, Matthew Finlayson, Oyvind Tafjord, Peter Clark, and Ashish Sabharwal. Increasing probability mass on answer choices does not always improve accuracy. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8392–8417, 2023.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Advances in Neural Information Processing Systems*, volume 36, pp. 78808–78823, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Brian Siyuan Zheng, Alisa Liu, Orevaoghene Ahia, Jonathan Hayase, Yejin Choi, and Noah A. Smith. Broken tokens? your language model can secretly handle non-canonical tokenizations. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 598–614, 2023.

A Related work

Tokenizer choice and tokenizer-free modeling. Modern language models usually rely on deterministic subword tokenization. The tokenizer affects sequence length, numerical representation, multilingual efficiency and inference cost, and character-level behavior (Mielke et al., 2021; Singh & Strouse, 2024; Petrov et al., 2023; Ahia et al., 2023; Zheng et al., 2025). One alternative is to remove subword tokenization altogether and model bytes or characters directly (Xue et al., 2022; Clark et al., 2022; Tay et al., 2022). Early byte-level models paid a large computational cost due to much longer sequences, so later work reduced the number of expensive model steps through downsampling or learned groups of bytes. MEGABYTE (Yu et al., 2023) and BLT (Pagnoni et al., 2025) take this further by processing bytes locally and running the main Transformer over larger patches instead. BLT in particular shows that this approach can match token-based models in FLOP-controlled experiments at scale, but it requires a different training and inference stack, and its reported FLOP savings have not yet been shown to translate into comparable wall-clock gains on the highly optimized systems used for standard token-based Transformers. Its downstream advantage is also not uniform across tasks. String Atlas keeps ordinary subword tokenization and the standard token-based Transformer architecture, but allow several tokenizers to be used with the same model.

Tokenizer design and vocabulary specialization. Another line of work asks how the tokenizer itself should be chosen. Studies of vocabulary size, compression, and tokenizer quality show that the best choice can depend on model scale, domain, language, and downstream task (Bostrom & Durrett, 2020; Tao et al., 2024; Goldman et al., 2024; Schmidt et al., 2024). Domain-specific and adaptive-vocabulary methods therefore replace, extend, or retrain vocabularies to better match a target domain or language (Dagan et al., 2024; Bommarito et al., 2025; Herold et al., 2025; Nakash et al., 2025; Liyanage & Yvon, 2026). These works motivate the domain, granularity, and algorithm variations we study. They still produce a model with one deployed tokenizer, whereas we look at whether several useful tokenizer choices can coexist in the same model.

Vocabulary overlap and multilingual sharing. Several multilingual studies provide evidence about how shared token identities affect transfer across languages. Chung et al. (2020) show that different language groups benefit from different vocabulary allocations, while Patil et al. (2022) find that favoring shared subwords can improve transfer to related low-resource languages. Limisiewicz et al. (2023) further show that the effect of overlap depends on the task, with shared identities helping some forms of lexical and semantic transfer but not universally. These results suggest that multilingual tokenizer routes may be a promising setting for String Atlas, although we do not evaluate multilingual tokenizers in this work.

Vocabulary adaptation after pretraining. Tokenizer-transfer methods instead start with a trained model and adapt it to a new vocabulary. They initialize the new lexical parameters by copying, averaging, aligning, predicting, or distilling from the original model, sometimes followed by continued training (Minixhofer et al., 2022; Dobler & de Melo, 2023; Ostendorff & Rehm, 2023; Remy et al., 2024; Minixhofer et al., 2024; Goddard & Fernandes Neto, 2025; Minixhofer et al., 2025). FOCUS (Dobler & de Melo, 2023) and ZeTT (Minixhofer et al., 2024), for example, provide ways to initialize a model under a tokenizer it was not originally trained with. This is a different setting from ours: the goal is to move a pretrained model from one tokenizer to another. String Atlas instead trains the tokenizer routes together so that the same model can use all of them directly.

Modular multilingual tokenizers. Concurrent work by Signe et al. (2026) also trains one model to operate with different tokenizer vocabularies, but in a more constrained multilingual/subtokenizer setting. They construct a large multilingual BPE or Unigram tokenizer from language-specific components, extract smaller subtokenizers for individual languages or language groups, and sample these subtokenizers during pretraining while restricting prediction to the active vocabulary. This allows the same model to use smaller language-specific vocabularies at inference, improving compression and reducing output-layer cost. Their subtokenizers are deliberately constructed as compatible subsets of one global tokenizer. String Atlas instead starts from independently constructed tokenizers and shares parameters when they contain the same string, allowing the supported tokenizers to differ in training data, vocabulary size, and even tokenization algorithm without first being combined into a common tokenizer.

Multiple segmentations of one text. Subword regularization, BPE-dropout, and StochasTok train a model on multiple segmentations of the same text (Kudo, 2018; Provilkov et al., 2020; Sims et al., 2026). These methods are especially relevant to our granularity experiments because they already expose the model to different token boundaries during training. The difference is that they vary the segmentation inside one vocabulary. String Atlas keeps several vocabularies and several next-token prediction spaces, while sharing rows only for tokens that represent the same byte string. Section 6 tests whether stochastic segmentation inside one vocabulary is enough, or whether directly training several tokenizer routes provides an additional benefit.

B PER-ROUTE GRANULARITY AND ALGORITHM RESULTS

The granularity and algorithm experiments in Section 6 each evaluate three tokenizer routes: 16k/32k/64k BPE for the two granularity settings, and BPE/WordPiece/Unigram for the fixed-size algorithm setting. The main table reports the mean gap over the three routes in each setting. Here we show the route-level gaps behind those means, which reveals additional structure in the failure modes of the other single-model baselines. As in Table 3, the first row reports the full-specialist BPB for each route, and the remaining rows report BPB gaps relative to that route's full specialist.

Table 6: Per-route quality in the granularity suites. Lower is better. The first row in each suite reports absolute BPB for full-budget specialists; all other numeric entries are BPB gaps relative to the full specialist on the same tokenizer route.

Suite	Method	BPE 16k	BPE 32k	BPE 64k	Average (↓)
Nested 16k/32k/64k	Specialists (full) [BPB]	1.1101	1.0860	1.0748	1.0903
	Specialists (CM)	+0.1122	+0.1079	+0.1099	+0.1100
	Private Tables	+0.0310	+0.1950	+0.1761	+0.1340
	Union-Longest	+0.2440	+0.0934	+0.0000	+0.1125
	StochasTok	+0.1473	+0.0805	+0.0380	+0.0886
	String Atlas (ours)	**+0.0132**	**+0.0161**	**+0.0262**	**+0.0185**
Independent 16k/32k/64k	Specialists (full) [BPB]	1.1038	1.0864	1.0735	1.0879
	Specialists (CM)	+0.1148	+0.1069	+0.1098	+0.1105
	Private Tables	+0.1314	+0.1269	+0.1307	+0.1297
	Union-Longest	+0.4025	+0.2056	+0.0946	+0.2342
	String Atlas (ours)	**+0.0213**	**+0.0167**	**+0.0285**	**+0.0222**

Granularity. In the nested suite, Union-Longest is exactly the native 64k specialist so its gaps on the 16k and 32k routes measure how well that trained 64k model can natively support the smaller tokenizers by constraining the vocabulary at inference time. The gap is zero on 64k, but rises to $+0.093$ on 32k and $+0.244$ on 16k; in other words, native performance at the largest granularity does not imply calibrated support for smaller routes. StochasTok shows a related but softer pattern, with its gap shrinking as the target route gets closer to the large-vocabulary view it is built around. String Atlas keeps all three nested routes close to their specialists by training those routes directly. The independent granularity suite removes prefix nesting, but we nonetheless observe similar results.

Table 7: Per-route quality in the algorithm suite. Lower is better. The first row reports absolute BPB for full-budget specialists; all other numeric entries are BPB gaps relative to the full specialist on the same tokenizer route.

Method	BPE 32k	WordPiece 32k	Unigram 32k	Average (↓)
Specialists (full) [BPB]	1.0843	1.0886	1.0922	1.0884
Specialists (CM)	+0.1082	+0.1102	+0.1131	+0.1105
Private Tables	+0.1230	+0.1188	+0.1389	+0.1269
Union-Longest	+0.1809	+0.1588	+0.4571	+0.2656
String Atlas (ours)	**+0.0278**	**+0.0225**	**+0.0301**	**+0.0268**

Algorithm. The String Atlas gaps are tightly clustered across BPE, WordPiece, and Unigram. This suggests that exact string sharing does not simply work by exploiting BPE-style nesting or a favorable granularity direction. It provides similar support across three distinct segmentation algorithms.

C STRESS TEST WITH MORE SKEWED DOMAIN TOKENIZERS

The main five-tokenizer suite in Section 5 uses domain-specific BPE tokenizers that share a large high-frequency core. This is the intended regime for String Atlas because the tokenizers are not identical, but they are also not adversarially constructed to have disjoint vocabularies. Still, this raises a natural question: does String Atlas only work because the main domain-specific tokenizers are especially easy to share across?

To stress this regime, we construct a more heavily skewed version of the five-tokenizer domain suite. We keep the same byte-level BPE recipe, vocabulary size, and tokenizer-training budget. We keep the general tokenizer the same as in the original suite, but retrain the four domain-specific tokenizers with more concentrated mixtures of their target domains (Table 8).

Table 8: Tokenizer-training mixtures for the more skewed domain-specific BPE suite. Each column is one tokenizer, and each row is a SlimPajama source-domain label. Entries are percentages of the 512 MiB tokenizer-training corpus for that tokenizer; darker shading indicates a larger mixture share. The general tokenizer is unchanged from the main suite.

	Tokenizer-training mixture				
Source domain	**General**	**Clean web**	**Code skew**	**Sci/math**	**Wiki/QA**
CommonCrawl	75%	20%	25%	25%	20%
C4	15%	70%	10%	10%	5%
GitHub	5%	0%	50%	0%	0%
StackExchange	1%	5%	15%	0%	35%
arXiv	3%	0%	0%	50%	0%
Wikipedia	1%	5%	0%	15%	40%

Compression specialization. The skewed tokenizers specialize more strongly than the main-suite tokenizers. Table 9 reports bytes per token on the same held-out SlimPajama source domains used in Table 1. As before, the goal of this table is not to evaluate language-model quality, but to verify that the tokenizer routes correspond to meaningfully different interfaces.

Table 9: Compression matrix for the more skewed domain-specific BPE suite. Each column is one tokenizer, and each row is an evaluation domain. Entries are bytes per token measured on held-out SlimPajama validation/test text from that domain. Higher is better, darker shading indicates better compression within each row, and bold marks the best tokenizer for that domain.

	Tokenizer (bytes/token ↑)				
Eval domain	**General**	**Clean web**	**Code skew**	**Sci/math**	**Wiki/QA**
CommonCrawl	**4.205**	4.174	4.042	4.071	3.991
C4	4.443	**4.466**	4.311	4.325	4.233
GitHub	3.101	2.975	**3.342**	2.773	3.122
StackExchange	3.274	3.258	**3.451**	3.091	3.371
arXiv	3.053	2.826	2.959	**3.193**	2.928
Wikipedia	2.975	3.209	2.825	3.408	**3.574**

Language-modeling results. Table 10 repeats the main comparison from Table 2 on the skewed suite. Even though the tokenizer routes are farther apart, String Atlas remains close to the full specialist ceiling, with a mean gap of $+0.0189$ BPB. This is only a modest increase from the main-suite gap, while the one-model baselines remain far from specialist quality.

Table 10: Main result on the more skewed five-tokenizer BPE suite. Specialists (full) are the five-run quality ceiling. All one-model methods use the same total pretraining budget. Gap is the quality tax vs. the full-specialist ceiling; lower BPB is better.

	Configuration			Performance	
Method	Total params	Lexical rows	Training FLOPs	Mean BPB (↓)	Gap (↓)
Specialists (full)	679.1M	160k	$4.270{\cdot}10^{18}$ (5×)	1.0916	–
Specialists (CM)	679.1M	160k	$0.852{\cdot}10^{18}$ (1×)	1.2917	+0.2002
Private Tables	332.5M	160k	$0.854{\cdot}10^{18}$ (1×)	1.3124	+0.2208
Union-Longest	177.1M	58.9k	$0.852{\cdot}10^{18}$ (1×)	1.3295	+0.2379
String Atlas (ours)	177.1M	58.9k	$0.852{\cdot}10^{18}$ (1×)	**1.1105**	**+0.0189**

Table 11: Per-route quality on the more skewed five-tokenizer BPE suite. The first row reports absolute BPB for full-budget specialists; all other numeric entries report BPB gaps relative to the full specialist on the same tokenizer route. Lower is better.

	Per-tokenizer performance (more skewed)					
Method	General	Clean web	Code skew	Sci/math	Wiki/QA	**Average** (↓)
Specialists (full) [BPB]	1.0832	1.0858	1.0958	1.0924	1.1007	1.0916
Specialists (CM)	+0.2044	+0.2013	+0.1975	+0.1998	+0.1978	+0.2002
Private Tables	+0.2232	+0.2172	+0.2164	+0.2213	+0.2258	+0.2208
Union-Longest	+0.1726	+0.1707	+0.2729	+0.2501	+0.3230	+0.2379
String Atlas (ours)	**+0.0199**	**+0.0176**	**+0.0190**	**+0.0200**	**+0.0180**	**+0.0189**

Zero-shot row transfer results. The skewed suite also makes the post-hoc row-transfer test from Section 7.3 substantially harder. Figure 6 shows that compared with the main suite, the off-diagonal BPB increases are larger (cell average: +0.1690 vs. +0.1015 in the main suite) and copied-token coverage is lower (cell average: 97.41% vs. 98.48% in the main suite).

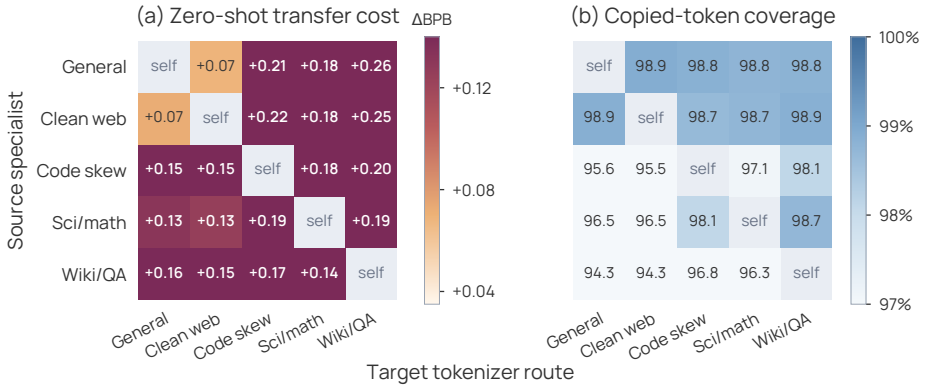


Figure 6: Zero-shot row transfer between specialists in the more skewed five-tokenizer BPE suite. Left: BPB increase relative to the target specialist. Right: copied target-position coverage before applying the contiguous-run filter.

Interpretation. Intuitively, making the domain tokenizers more specialized makes post-hoc row transfer substantially less reliable. Fewer target positions are covered by copied rows, and even the rows that can be copied are less well calibrated for the target tokenizer. String Atlas trains under each route jointly, so it is much less affected, and its mean gap to the specialist ceiling increases only slightly to $+0.0189$ BPB. This suggests that the strong results in the main suite are not simply a consequence of using unusually similar tokenizers. String Atlas still depends on the tokenizers sharing some strings, but it continues to work well even when their vocabularies are pushed further toward their respective domains.

D UNION-LONGEST DETAILS

Union-Longest uses the same pooled vocabulary that String Atlas builds, but turns it into one ordinary tokenizer instead of keeping the original tokenizer routes. For the five domain-specific BPE tokenizers, String Atlas first forms an atlas by taking the union of all emitted byte strings; this gives 50,184 unique strings. Union-Longest uses that same set of 50,184 strings as a single vocabulary, and uses greedy maximum matching to segment text. At each byte position, it finds all vocabulary strings that match the remaining byte prefix, picks the longest one, advances by that many bytes, and repeats. Thus Union-Longest has access to exactly the same byte strings as String Atlas, but it discards the original BPE merge rules and replaces the five tokenizer routes with one longest-match route.

The main-body comparison uses a compute-matched Union-Longest checkpoint. Union-Longest predicts over 50,184 rows, rather than the 32k rows active under each original BPE route, so an 8k-step Union-Longest run uses more training compute under the active-parameter proxy used in Table 2. We therefore stop it at 6,619 steps for the main compute-matched baseline. For completeness, we also train the same model for the full 8k steps and report both versions here.

Table 12: Union-Longest with and without compute matching. FLOPs use the same active-parameter training proxy as Table 2. The native route evaluates the longest-match tokenizer itself. The route-restricted mean evaluates the same model under the five original BPE tokenizer routes by mapping each original token to the union row with the same emitted byte string and restricting the softmax to that route's mapped rows.

Union-Longest run	Steps	Train tokens	Train FLOPs	Native BPB	Route-restricted mean
Compute-matched	6,619	867M	$0.852 \cdot 10^{18}$	1.0991	1.2835
Full	8,000	1.048B	$1.030 \cdot 10^{18}$	1.0821	1.2688

The full Union-Longest run shows that compute matching is not the main issue. Training for the full 8k steps improves native longest-match BPB from 1.0991 to 1.0821, but the route-restricted mean remains high (1.2688). Thus Union-Longest is a strong tokenizer for its own segmentation, but extra training does not make it a good substitute for the five original BPE tokenizers.

This raises the complementary question: if longest-match is useful as its own tokenizer, can String Atlas simply include it as another route? We test this by adding Union-Longest as a sixth route while keeping the same atlas rows, since its vocabulary is exactly the union already used by String Atlas.

Figure 7 summarizes the asymmetry. Union-Longest reaches 1.0821 BPB on its own longest-match route with the full 8k-step run, but the same weights give a much worse 1.2688 mean BPB when evaluated through the five original BPE routes. A five-route String Atlas model has the opposite limitation: it is strong on the original five routes, but is not automatically trained for longest-match. Once longest-match is added during training, String Atlas reaches 1.1093 BPB on that route while keeping the original five-route mean essentially unchanged (1.1021 vs. 1.1027). The takeaway is that having all of the strings in one vocabulary is not enough by itself. Union-Longest works well on its own longest-match tokenizer, but does not replace the five original tokenizers; String Atlas can instead add longest-match as another route while preserving quality on the original five.

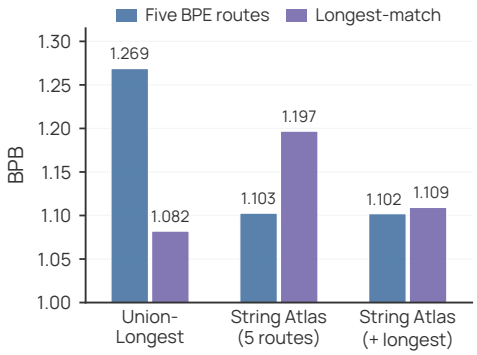


Figure 7: Adding Union-Longest as a sixth String Atlas route. Bars compare BPB on the mean of the five original BPE tokenizer routes and on the longest-match tokenizer.

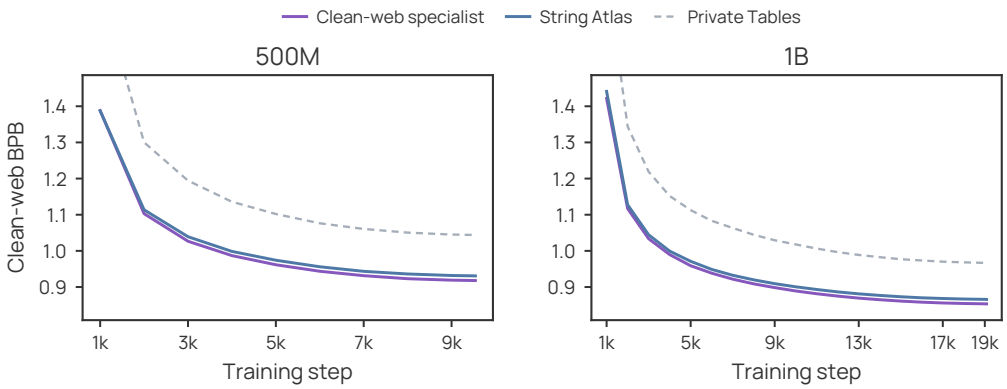


Figure 8: Clean-web BPB over training steps for the larger scale models. Private Tables learns much more slowly than String Atlas. String Atlas stays close to the same-scale clean-web specialist while additionally supporting the four other routes at the same time.

E LARGER-SCALE RUNS

We use the 86.6M-parameter trunk in our main experiments because it makes the full experimental grid feasible (all five specialists, the one-model baselines, and the mechanism ablations can be trained in the same suite). To check that the pattern is not specific to this small-trunk regime, we repeat the main five-tokenizer setting at larger model sizes with Chinchilla-style pretraining budgets (Hoffmann et al., 2022). At both 500M and 1B scale, we run all five tokenizer routes for the full and compute-matched specialists, Private Tables, Union-Longest, and String Atlas. This lets us test whether String Atlas's near-specialist quality and advantage over the one-model baselines persist across model scales and tokenizer routes.

Table 13: Per-route BPB gaps for the larger-scale experiments on the five domain-specific BPE tokenizers. The first row reports absolute BPB for full-budget specialists; all other entries report BPB gaps relative to the full specialist on the same tokenizer route. Lower is better.

		Per-tokenizer performance					
Trunk	**Method**	General	Clean web	Code skew	Sci/math	Wiki/QA	**Average** (↓)
500M	Specialists (full) [BPB]	0.9190	0.9179	0.9201	0.9206	0.9215	0.9198
	Specialists (CM)	+0.1935	+0.1955	+0.1964	+0.1984	+0.1959	+0.1959
	Private Tables	+0.1267	+0.1260	+0.1282	+0.1290	+0.1262	+0.1272
	Union-Longest	+0.1733	+0.1733	+0.2224	+0.2105	+0.2362	+0.2031
	String Atlas (ours)	**+0.0117**	**+0.0129**	**+0.0144**	**+0.0132**	**+0.0125**	**+0.0129**
1B	Specialists (full) [BPB]	0.8535	0.8534	0.8547	0.8550	0.8559	0.8545
	Specialists (CM)	+0.1421	+0.1436	+0.1419	+0.1427	+0.1433	+0.1427
	Private Tables	+0.1120	+0.1134	+0.1143	+0.1139	+0.1141	+0.1135
	Union-Longest	+0.1803	+0.1793	+0.2282	+0.2168	+0.2412	+0.2092
	String Atlas (ours)	**+0.0121**	**+0.0123**	**+0.0136**	**+0.0131**	**+0.0121**	**+0.0126**

Discussion. At 500M scale, String Atlas is within $+0.0129$ BPB of the specialist fleet on average, with per-route gaps ranging from $+0.0117$ to $+0.0144$. The other one-model configurations remain much farther from the ceiling: Private Tables has a $+0.1272$ mean gap and Union-Longest a $+0.2031$ gap, while the compute-matched specialist fleet is $+0.1959$ behind. The same pattern holds at 1B scale. String Atlas has a $+0.0126$ mean gap, with every route between $+0.0121$ and $+0.0136$, compared with $+0.1135$ for Private Tables, $+0.2092$ for Union-Longest, and $+0.1427$ for compute-matched specialists. Thus String Atlas's near-specialist quality persists across all five routes as the trunk and training budget scale up; Figure 8 illustrates the corresponding optimization behavior on the clean-web route for both model scales.

F POST-HOC TOKENIZER TRANSFER AND DOWNSTREAM EVALUATION

The main experiments compare methods that are trained from the start to support the tokenizer routes being evaluated. In this section, we add two complementary checks. First, we compare against FOCUS (Dobler & de Melo, 2023), which instead starts from a finished specialist and transfers it to a new tokenizer. Second, we evaluate the models on seven downstream multiple-choice benchmarks to check whether the differences in language-model likelihood are reflected in task performance.

F.1 ZERO-SHOT TRANSFER WITH FOCUS

FOCUS provides a stronger post-hoc alternative to simply copying the rows that two tokenizers have in common. Starting from a trained source model, it copies rows for tokens shared with the target vocabulary and constructs the remaining target rows from auxiliary token representations. We evaluate FOCUS strictly zero-shot; i.e., after constructing the new lexical rows for the target tokenizer, we make no further updates to the source language model. For the five domain-specific tokenizers, the source is the full General specialist. As such, the General route itself is evaluated natively, while Clean web, Code skew, Sci/math, and Wiki/QA are evaluated each with their own newly initialized embedding and unembedding tables.

For each target tokenizer, we train the auxiliary token-level FastText model used by FOCUS on 1B tokens from the same FineWeb-Edu training stream used for language-model pretraining. We use the same FOCUS hyperparameters reported by Dobler & de Melo (2023): CBOW with 300-dimensional vectors, three epochs, ten negative samples, and a minimum count of 10. For overlapping strings, we copy the embedding and unembedding rows directly from the source model. For a target token that does not overlap with the source vocabulary, FOCUS uses similarities in the auxiliary FastText space to form a sparse weighted combination of overlapping source rows. Embedding and unembedding tables are initialized independently in this way.

Because FOCUS is a post-hoc transfer procedure that requires training an auxiliary FastText model for each target tokenizer, we do not include it in the compute-matched accounting of the main baselines. Our goal here is simply to study: *once a source specialist has been trained, how close can FOCUS get to the specialists for other tokenizer routes without any additional language-model training?*

We repeat this experiment across the tokenizer suites in the paper. The General specialist is the source for both domain-specific suites, the 16k specialist is the source for the two granularity suites, and the BPE specialist is the source for the BPE/WordPiece/Unigram suite. Table 14 summarizes the results. Because the source route is evaluated natively and therefore has zero transfer gap, we report both the mean over all routes and the mean over only the routes that actually require transfer.

Table 14: Zero-shot tokenizer transfer with FOCUS. BPB gaps are measured relative to the full specialist for each route; lower is better. For FOCUS, "All" includes the native source route, where the gap is zero. "Transferred" averages only the target routes that require tokenizer transfer. String Atlas is trained jointly on all routes, so we report its ordinary all-route mean here.

			FOCUS		
Suite	**Scale**	**Source**	**All**	**Transferred**	**String Atlas**
Domain BPE	86M	General	+0.0591	+0.0739	**+0.0137**
	500M	General	+0.0639	+0.0799	**+0.0129**
	1B	General	+0.0658	+0.0823	**+0.0126**
Skewed domain BPE	86M	General	+0.0974	+0.1218	**+0.0189**
Nested 16k/32k/64k	86M	16k	+0.1131	+0.1697	**+0.0185**
Independent 16k/32k/64k	86M	16k	+0.1150	+0.1726	**+0.0222**
BPE/WordPiece/Unigram	86M	BPE	+0.3383	+0.5074	**+0.0268**

Results. FOCUS is substantially stronger than the simpler post-hoc alternatives, but still leaves a clear gap to full specialists. On the main domain suite, its transferred-route gap is about $+0.07$ BPB at 86M and remains around $+0.08$ as the model scales to 500M and 1B. In contrast, String Atlas stays near $+0.01$ BPB while supporting all five routes directly.

Notably, FOCUS's BPB gap also grows much more substantially than String Atlas's as the target tokenizer moves farther from the source, from a $+0.12$ transferred-route gap on the more skewed domain tokenizers, to a roughly $+0.17$ on both granularity suites. Cross-algorithm transfer is even more difficult, with FOCUS producing gaps of $+0.5090$ and $+0.5058$ BPB for WordPiece and Unigram, respectively. We suspect this is because changing the tokenization algorithm alters both (1) which lexical units are available and (2) how those units are *composed into a segmentation*, so the source model's learned rows and boundary patterns transfer less cleanly than they do across BPE tokenizers that differ only in domain or vocabulary size.

F.2 Downstream multiple-choice evaluation

Our main metric throughout the paper is BPB because it directly measures the quality of the language model and remains comparable across tokenizers. We also evaluate whether the same pattern appears on downstream multiple-choice tasks.

We evaluate on HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), and SciQ (Welbl et al., 2017) through `lm-eval` v0.4.12 (Gao et al., 2023). All evaluations are zero-shot and use the standard task prompts provided by `lm-eval`. For each model and for each benchmark, we evaluate every tokenizer route separately and average their scores to obtain the performance of that model on that benchmark. We then average these seven benchmark scores to obtain the overall downstream result.

Additionally, we note that multiple-choice accuracy only depends on which candidate receives the highest score, and it can hide changes in the absolute likelihood assigned to the answers (Wiegreffe et al., 2023). To measure this directly, we also report the *answer NLL gap*. For candidate answer a under route r, we compute

$$\Delta_{\mathrm{NLL}}(a) = \frac{\log p_{\mathrm{spec},r}(a \mid x) - \log p_{\mathrm{method},r}(a \mid x)}{|\mathrm{bytes}(a)|}, \tag{5}$$

where $p_{\mathrm{spec},r}$ is the full specialist trained for the same tokenizer route. The value is measured in nats per answer byte, and more positive values correspond to the method assigning lower likelihood to the answer than the matching full specialist.

We compute this quantity for every answer, not only the correct one. We first average candidates within each task, then average the seven tasks equally, and finally average tokenizer routes equally. Accuracy can remain almost unchanged even after the model has lost substantial likelihood across the answer choices, so we use the NLL gap to probe that loss and to provide a more complete picture of model performance.

Table 15: Downstream multiple-choice evaluation on the *five domain-specific tokenizers suite* across three scales. Accuracy is the macro-average over HellaSwag, PIQA, WinoGrande, ARC-Easy, ARC-Challenge, OpenBookQA, and SciQ; higher is better. Answer NLL gap (Δ_{NLL}) is measured relative to the full specialist on the same tokenizer route; lower is better.

	86M		**500M**		**1B**	
Method	Acc. ↑	Δ_{NLL} ↓	Acc. ↑	Δ_{NLL} ↓	Acc. ↑	Δ_{NLL} ↓
Specialists (full)	0.4081	0.0000	0.4542	0.0000	0.4883	0.0000
Specialists (CM)	0.3726	+0.1385	0.4021	+0.1059	0.4311	+0.0780
Private Tables	0.3724	+0.1324	0.4159	+0.0794	0.4362	+0.0672
Union-Longest	0.3914	+0.2168	0.4284	+0.2064	0.4527	+0.2134
FOCUS	0.3994	+0.0537	0.4417	+0.0661	0.4699	+0.0692
String Atlas	**0.4067**	**+0.0161**	**0.4559**	**+0.0197**	**0.4819**	**+0.0181**

Table 15 reports the main five-tokenizer suite across model scales. The downstream results follow the same broad pattern as the language-modeling evaluation. String Atlas remains close to the full specialists at all three scales. The accuracy differences are relatively small, especially for FOCUS, but the answer NLL gap reveals a clearer separation between the methods.

Table 16: Downstream evaluation across the remaining 86M tokenizer suites. Metrics and aggregation follow Table 15. StochasTok is evaluated only for the nested granularity suite.

	Skewed Domain		Nested 16k+		Indep. 16k+		BPE/WP/Uni	
Method	Acc. ↑	Δ_{NLL} ↓	Acc. ↑	Δ_{NLL} ↓	Acc. ↑	Δ_{NLL} ↓	Acc. ↑	Δ_{NLL} ↓
Specialists (full)	0.4066	0.0000	0.4081	0.0000	0.4058	0.0000	0.4080	0.0000
Specialists (CM)	0.3732	+0.1319	0.3866	+0.0601	0.3851	+0.0669	0.3858	+0.0703
Private Tables	0.3713	+0.1407	0.3804	+0.0880	0.3822	+0.0763	0.3817	+0.0918
Union-Longest	0.3920	+0.2910	0.4016	+0.1192	0.3902	+0.2402	0.3880	+0.2567
StochasTok	–	–	0.4015	+0.0826	–	–	–	–
FOCUS	0.4003	+0.0943	0.3817	+0.0786	0.3838	+0.0852	0.3824	+0.2926
String Atlas	**0.4067**	**+0.0113**	**0.4017**	**+0.0118**	**0.4001**	**+0.0133**	**0.4061**	**+0.0177**

The same comparison is shown for the harder tokenizer suites in Table 16. Again, accuracy compresses many of the differences between methods. On the BPE/WordPiece/Unigram suite, for example, FOCUS reaches an accuracy of 0.3824 compared with 0.4061 for String Atlas, while its answer NLL gap rises to $+0.2926$ compared with only $+0.0177$ for String Atlas. Similar, though smaller, differences appear in the domain and granularity suites.

Overall, the downstream results agree with the likelihood-based evaluation but also support our choice of likelihood/BPB as the primary metric in the paper. At these model scales, multiple-choice accuracy is relatively coarse, and substantial changes in the probability assigned to the answer choices do not always change which choice is ranked first. The answer NLL gap makes those changes visible, and under this more sensitive measure String Atlas remains much closer to the route-matched full specialists across every tokenizer suite.

G Experimental details

G.1 Tokenizer suite construction

Table 17: Tokenizer-training mixtures for the five domain-specific BPE tokenizers used in the main experiments. Each column is one tokenizer, and each row is a SlimPajama source-domain label. Entries are percentages of the 512 MiB tokenizer-training corpus for that tokenizer; darker shading indicates a larger mixture share.

	Tokenizer-training mixture				
Source domain	**General**	**Clean web**	**Code skew**	**Sci/math**	**Wiki/QA**
CommonCrawl	75%	50%	45%	45%	45%
C4	15%	40%	15%	15%	15%
GitHub	5%	0%	30%	0%	0%
StackExchange	1%	5%	10%	0%	15%
arXiv	3%	0%	0%	30%	0%
Wikipedia	1%	5%	0%	10%	25%

Domain-specific BPE suite. The main five-tokenizer suite uses byte-level BPE throughout. Each tokenizer has target vocabulary size 32k, including the single `<unk>` special token, and uses the same byte-level pre-tokenizer, byte fallback, initial byte alphabet, minimum frequency 2, and `<unk>` handling. The only changing ingredient is the tokenizer-training corpus. For each tokenizer, we sample 512 MiB of SlimPajama training text according to the mixture in Table 17. The language models are then trained and evaluated on the separate FineWeb-Edu stream used in the main experiments.

Nested granularity suite. The nested granularity suite uses the general-mixture recipe from Table 17. We train byte-level BPE tokenizers with requested vocabulary sizes 16k, 32k, and 64k on the same 512 MiB general-mixture corpus, using the same special token and BPE trainer settings

as above. The resulting tokenizer files are prefix-nested: the 16k vocabulary and merge list are exact prefixes of the 32k tokenizer, and the 32k vocabulary and merge list are exact prefixes of the 64k tokenizer. Thus the smaller tokenizers are materialized tokenizer files with their own active vocabularies, while preserving exact merge-rank nesting inside the larger vocabulary.

Independent granularity suite. The independent granularity suite uses the same general-mixture source and the same byte-level BPE settings, but trains each vocabulary size on its own 512 MiB chunk. The 16k tokenizer uses the first chunk, matching the nested and algorithm-suite tokenizer-training text; the 32k and 64k tokenizers use the next two disjoint chunks. This keeps the mixture and trainer fixed while making the three vocabularies independently learned, so the 16k, 32k, and 64k tokenizers no longer form prefix vocabularies.

BPE, WordPiece, and Unigram suite. The algorithm suite trains BPE, WordPiece, and Unigram tokenizers on the same first 512 MiB general-mixture chunk. All three request 32k vocabulary entries, including `<unk>`, and use byte-level pre-tokenization so that pieces can be mapped back to byte strings. BPE uses the same byte-level BPE trainer as the other suites. WordPiece uses the same byte-level pre-tokenizer with `##` as its continuation prefix. Unigram uses a byte-level Unigram trainer with byte fallback.

Atlas keys for non-BPE tokenizers. For BPE and Unigram pieces, the atlas key is the emitted byte string obtained by reversing the byte-level alphabet. WordPiece needs one additional boundary bit: a token such as `##ing` emits the bytes for `ing`, but it is only valid as a continuation piece. In the algorithm suite, ordinary WordPiece tokens are keyed by their emitted bytes, while continuation WordPiece tokens are keyed by those emitted bytes plus a continuation marker. This avoids collisions between WordPiece entries that emit the same bytes in different boundary states, while still allowing rows to be shared when the emitted string and boundary state are compatible.

G.2 Architectural details

All models use a decoder-only Transformer with learned absolute positional embeddings, pre-LayerNorm blocks, GELU activations, and untied input and output embeddings. Unless otherwise stated, the context length is 2048 tokens and dropout is set to 0.1. The 86.6M model is the main experimental setting, while the 500M and 1B models are used for the larger-scale checks in Appendix E. The parameter names refer to the shared Transformer trunk; the total parameter counts of multi-tokenizer models additionally depend on whether lexical rows are private, unioned, or shared through the atlas.

Table 18: Decoder architecture and pretraining configuration.

	86.6M	**500M**	**1B**
Layers	12	18	20
d_{model}	768	1536	2048
MLP dim.	3072	6144	8192
Attention heads	12	24	32
KV heads	12	24	32
Head dim.	64	64	64
Context length	2048	2048	2048
Dropout	0.1	0.1	0.1
Training tokens	1.05B	10.00B	20.00B
Optimizer	AdamW	AdamW	AdamW
Peak LR	6e-4	3e-4	2e-4
Weight decay	0.1	0.1	0.1
Grad. clipping	1.0	1.0	1.0
Global batch	64	512	512
Optimizer steps	8,000	9,542	19,083

All language models are pretrained on FineWeb-Edu (Penedo et al., 2024). We use AdamW with weight decay 0.1 and clip gradients to norm 1.0. The learning rate uses linear warmup followed by cosine decay to 10^{-5}. The 86.6M runs warm up for 26.2M tokens, corresponding to 200 optimizer steps, while the 500M and 1B runs warm up for 1.05B tokens, corresponding to 1000 optimizer steps. The token budgets, peak learning rates, batch sizes, and total optimizer steps are summarized in Table 18.

Configuration accounting. Table 19 summarizes the resource accounting for the 86.6M-scale experiments. The full-specialist setting trains one complete model per tokenizer route and therefore gives the quality ceiling at an $R\times$ training budget. The compute-matched specialist setting keeps the same separate-model structure, but divides the one-model budget across the R specialists. Private Tables and String Atlas both train directly on all original tokenizer routes using one shared

Transformer trunk; they differ only in whether lexical rows remain tokenizer-local or are shared by emitted string.

Union-Longest is different in that it trains a single deterministic tokenizer built from the union of emitted strings, rather than training on the original tokenizer routes. We still evaluate it through the original routes by mapping each route's tokens to the corresponding union rows and restricting the prediction space to that route. StochasTok appears only in the nested granularity suite and uses one 64k vocabulary with stochastic segmentations, rather than separate objectives for the 16k, 32k, and 64k routes.

Table 19: Experimental configurations at 86.6M scale. Parameter counts include the Transformer trunk(s), untied input and output lexical tables, and output biases. Specialist counts are totals over the whole fleet. Budget is measured relative to one full single-model pretraining run.

Suite	Configuration	Routes trained	Trunks	Rows	Params	Budget
Domain BPE 32k	Specialists (full)	5	5	160k	679.1M	5×
	Specialists (CM)	5	5	160k	679.1M	1×
	Private Tables	5	1	160k	332.5M	1×
	Union-Longest	1	1	50.2k	163.8M	1×
	String Atlas	5	1	50.2k	163.8M	1×
Nested 16k/32k/64k	Specialists (full)	3	3	112k	432.0M	3×
	Specialists (CM)	3	3	112k	432.0M	1×
	Private Tables	3	1	112k	258.8M	1×
	Union-Longest	1	1	64k	185.0M	1×
	StochasTok	1	1	64k	185.0M	1×
	String Atlas	3	1	64k	185.0M	1×
Independent 16k/32k/64k	Specialists (full)	3	3	112k	432.0M	3×
	Specialists (CM)	3	3	112k	432.0M	1×
	Private Tables	3	1	112k	258.8M	1×
	Union-Longest	1	1	65.1k	186.6M	1×
	String Atlas	3	1	65.1k	186.6M	1×
BPE/WordPiece/Unigram 32k	Specialists (full)	3	3	96k	407.4M	3×
	Specialists (CM)	3	3	96k	407.4M	1×
	Private Tables	3	1	96k	234.2M	1×
	Union-Longest	1	1	57.3k	174.7M	1×
	String Atlas	3	1	57.3k	174.7M	1×

G.3 Building the String Atlas

Given an ordered set of tokenizer routes, we build the atlas once before training. For every tokenizer T_r and every token id $i \in V_r$, we compute the atlas key $\kappa_r(i)$ described above. Normally this is just the emitted byte string, but for WordPiece continuation pieces specifically, we use the emitted byte string together with the continuation marker. We then union these keys into one global set

$$S = \{\kappa_r(i) : r \in \{1, \dots, R\},\ i \in V_r\}. \tag{6}$$

We also stores a route-local lookup table $A_r \in \{1, \dots, |S|\}^{|V_r|}$, where $A_r[i]$ is the atlas row used by token id i under route r. These lookup tables are fixed after construction. The trainable lexical parameters are one input embedding row $E[s]$, one output unembedding row $U[s]$, and one output bias $b[s]$ for each atlas key $s \in S$.

This construction keeps tokenizer behavior separate from row ownership. The tokenizer still decides how raw text is segmented and which ids are valid next tokens under route r; the atlas only decides which input and output rows those ids use. If two routes emit the same atlas key, their ids point to the same row. If a string appears in only one route, it keeps its own row in the same shared table.

G.4 TRAINING AND INFERENCE

The implementation may cache tokenized streams for efficiency, but the training objective is equivalent to the route-sampling procedure in Algorithm 1.

Algorithm 1 Pretraining with a String Atlas model

Require: Tokenizers $T_1, \dots, T_R$ with vocabularies $V_1, \dots, V_R$; route distribution q; raw-text training stream $\mathcal{D}$

Build atlas S and lookup maps $A_1, \dots, A_R$
Initialize the shared Transformer trunk and atlas rows E, U, b
for each optimizer step **do**
  $\mathcal{L} \leftarrow 0$
  for each accumulated microbatch **do**
    Sample route $r \sim q$
    Draw raw text windows from $\mathcal{D}$
    Tokenize each window with T_r to obtain ids $i_1, \dots, i_n$
    Embed each input id with $E[A_r[i_t]]$ and run the shared trunk to produce h_t
    for each position t **do**
      for all $j \in V_r$ **do**
        $\ell_t(j) \leftarrow h_t^\top U[A_r[j]] + b[A_r[j]]$
      end for
      $\mathcal{L} \leftarrow \mathcal{L} - \log \frac{\exp \ell_t(i_{t+1})}{\sum_{j \in V_r} \exp \ell_t(j)}$
    end for
  end for
  Update the trunk and atlas rows using $\nabla \mathcal{L}$
end for

For the main multi-tokenizer runs, the route distribution is uniform over the routes in the suite, so the one-model training budget is spread across the available tokenizers. The loss is otherwise the standard next-token cross-entropy for the sampled tokenizer. Thus a shared row receives target-event updates whenever any route emits its string, while the normalizer remains the active vocabulary of the route that produced the batch.

At inference time, a route is chosen first. The prompt is tokenized with that route's tokenizer, input ids are embedded through the corresponding atlas map A_r, and each next-token distribution is normalized over V_r using the logits above. Generated ids are decoded by the same tokenizer T_r. The model therefore behaves like an ordinary language model under the selected tokenizer, with no cross-route softmax or marginalization required. To compare or route among tokenizers at deployment time, one runs the model under those routes separately or chooses a route before decoding.